UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Autonomous Systems

Course Project - A.Y. 2023-24

Professors:
Giuseppe Notarstefano
Ivano Notarnicola

Students:
Gabriele Ruscelli
Giuseppe Speciale
Francesco D. Bossio

Academic year 2023/2024

Abstract

In this project we are asked to work on two tasks in a distributed setting; the first is to build a distributed classifier based on a logistic function and to test its performances with a test set, while in the second the aim is instead on aggregative optimization to control a multi-robot system towards some targets while maintaining the formation tight.

Contents

Introduction

The project consists of two main tasks.

The first one involves a data analytics application an is composed of three sub-tasks:

- *1.1 Distributed Optimization*: in which the aim is to get a network of agents work togheter in a distributed fashion on a simple problem.
- *1.2 Centralized Classification*: build a classifier in a centralized framework and test that everything works as expected.
- *1.3 Distributed Classification*: merge the two tasks before and get the network of agents to work on a difficult problem.

The second task deals with the control for multi-robot systems in ROS 2.

- *2.1 Problem Setup*: here the objective is to move the agents to some targets (moving or not) while maintaining the formation tight.
- *2.2 ROS2 Implementation*: now move the previous task in a ROS2 environment and visualize the behaviour of the agents.
- *2.3 Moving Inside a Corridor*: finally make the agents reach some targets through a corridor using a barrier function.

Chapter 1

Task 1: Distributed Classification via Logistic Regression

Suppose to have a set of N agents $I = \{1, .., N\}$ that want to cooperatively determine a nonlinear classifier for a set of points in a given feature space.

1.1 Task 1.1: Distributed Optimization

1.1.1 Problem description and theory recall

As requested by the task we have implemented the Gradient Tracking algorithm and run different simulations to test its effectiveness on a unconstrained quadratic optimization problem.

$$\min_z \sum_{i=1}^{N} \ell_i(z) = \sum_{i=1}^{N} z^T Q_i z + z^T R_i$$

with $z \in \mathbb{R}^d$ while $\ell_i : \mathbb{R}^d \to \mathbb{R}$ is a quadratic function $\forall i \in \{1, \ldots, N\}$ with Q_i strictly positive and symmetric matrix. Since the unconstrained quadratic optimization problem admits closed form solution we can compute the optimal solution of the problem setting the gradient of the cost function to zero $\nabla \ell(z^*) = 0$ thus

$$z^* = -(\sum_{i=1}^{N} Q_i)^{-1} (\sum_{i=1}^{N} R_i)$$

In this task the N agents are cooperatively working to determine the optimal solution. The typology of the communication graph $G = (\{1, .., N\}, E)$ is defined by the set of edges E in the graph, we tested on our algorithm different typology of graphs: completely connected graph, cycle graph, path graph and binomial graph. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ is defined such that its elements a_{ij} are weighted by the Metropolis Hastings method:

$$A_{ij} = \begin{cases} \frac{1}{1+\max\{d_i,d_j\}} & \text{if } (i,j) \in E \text{ and } i \neq j \\ 1 - \sum_{h \in N_i \setminus \{i\}} A_{ih} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where:

- d_i is the degree of vertex i.
- N_i is the set of neighbors of vertex i.

This method guarantees that A is doubly stochastic and symmetric matrix.

The described problem is solved in a distributed setting by means of the gradient tracking algorithm.

$$\begin{cases} z_i^{k+1} = \sum_{j \in N_i} a_{ij} z_j^k - \alpha s_i^k, & z_i^0 \in \mathbb{R}^d, \\ s_i^{k+1} = \sum_{j \in N_i} a_{ij} s_j^k + \nabla_i \ell(z_i^{k+1}) - \nabla_i \ell(z_i^k), & s_i^0 = \nabla \ell_i(z_i^0) \end{cases}$$

Since G is a connected and aperiodic indirect graph, A is doubly stochastic, the cost function of each agent is strongly convex and has a Lipschitz continuous gradient.

Then the use of gradient tracking algorithm guarantees the convergence of z_i^k to z^* for $k \to \infty \forall i \in \{1, ..., N\}$

1.1.2 Results

In this section we show the performances of our algorithm applied to different type of graphs. The chosen parameters are the following:

- $N = 5$ number of agents.
- $d = 2$ dimension of the agent's state.
- $\alpha = 10^{-3}$ constant step size.
- $TerminalCondition = 10^{-4}$ termination condition.

We implemented a gradient tracking algorithm that is performed until the computed gradient of each agent s_i^{k+1} is less then a certain $TerminalCondition$, thus the iteration k is the last iteration if

$$s_i^{k+1} <= TerminalCondition \forall i \in \{1, ..., N\}$$

General results

In order to show the performances of the algorithm we evaluate the norm of the global gradient of the agents $|\nabla \ell(z^k)|$, the error between the cost of the algorithm and the cost of the optimal solution $\sum_{i=1}^{N} |\ell(z^k) - \ell(z^*)|$ and the error between the optimal solution and the solution of the gradient tracking $\sum_{i=1}^{N} ||z_i^k - z^*||$ along the iterations k .

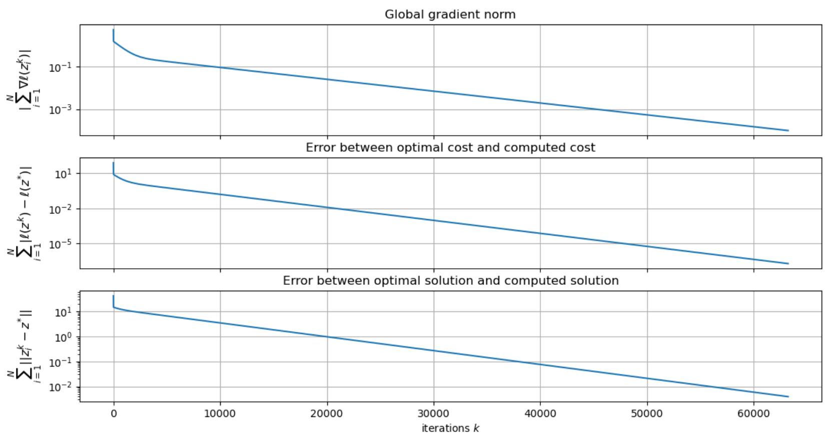


Figure 1.1: Results of the algorithm using path graph

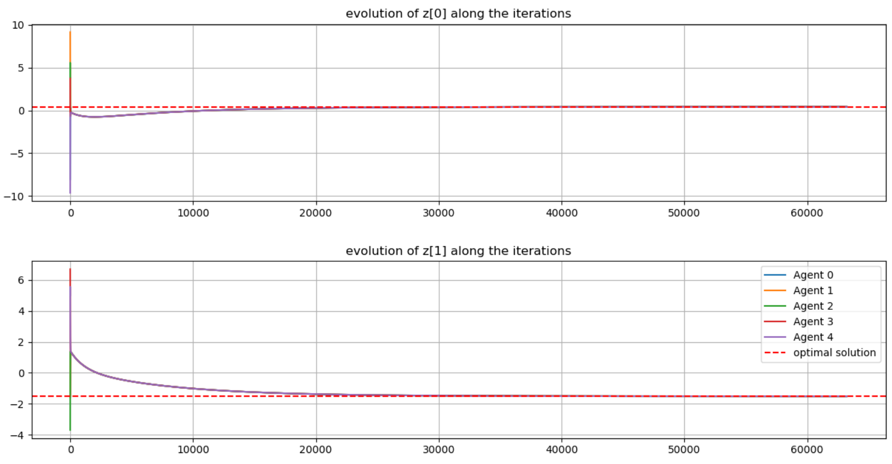
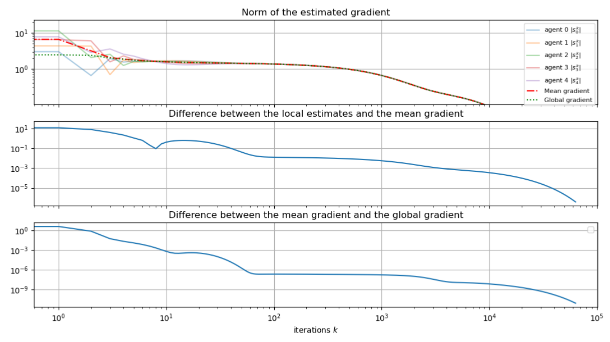


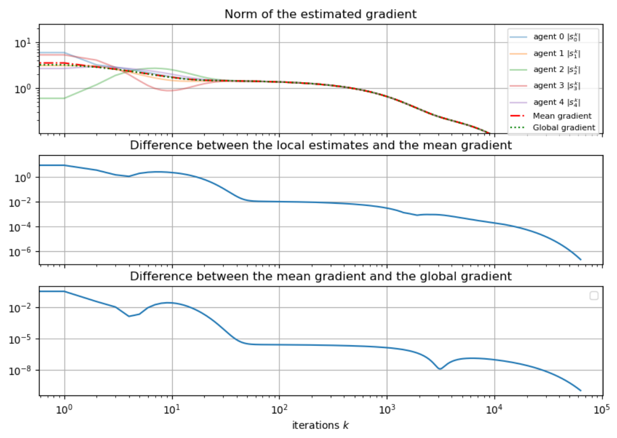
Figure 1.2: Evolution of the states of the agents using path graph

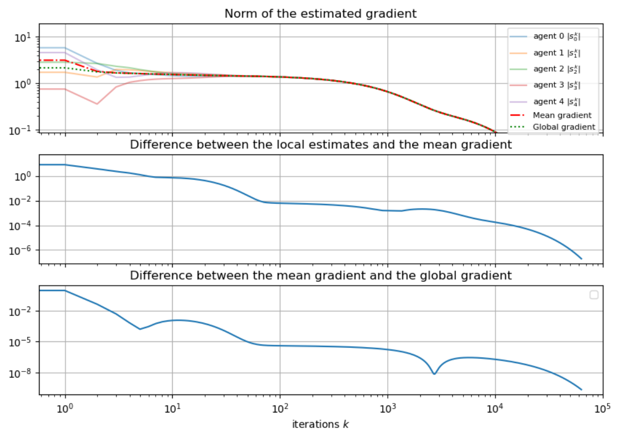
The general result is that independently from which type of graph we chose to use is that the algorithm converges to an unique solution and both the norm of the gradient and the error between the optimal solution and its cost tends to 0.

Differences between different type of graphs

From figure 1.1 we cannot appreciate the behaviour between different types of graph, for this purpose we show the evolution of the norm of the estimated gradient $|s_i^k|$ with respect to the global gradient $|\nabla\ell(z^k)|$. To evaluate if the agents reached consensus the idea we had is to compute the distance between the mean of the norm of the estimated gradient $\sum_{i=1}^N \frac{1}{N}|s_i^k|$ and each $|s_i^k|$, thus $\sum_{i=1}^N(\sum_{j=1}^N(\frac{1}{N}|s_j^k|) - |s_i^k|)$ Finally we computed the distance between the mean norm of the estimated gradient, i.e. $\sum_{i=1}^N \frac{1}{N}|s_i^k|$ with respect to the norm of the global gradient $|\nabla\ell(z^k)|$.
From these following plots we can better appreciate each different behaviours for each type of graph.







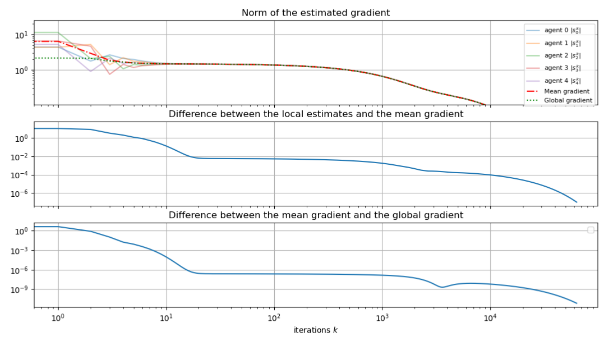


Figure 1.3: From the upper left: Path, Star, Binomial (p = 0.3), and Cycle graph.

1.2 Task 1.2: Centralized Classification

The task is now to generate a dataset of points in the space and label them by separating the set with a nonlinear function.

$$\{x \in \mathbb{R}^d \mid w^\top \phi(x) + b = 0\}$$

with $\phi : \mathbb{R}^d \to \mathbb{R}^q$ nonlinear function and $w \in \mathbb{R}^q, b \in \mathbb{R}$ parameters. The dataset of $\mathcal{M} \in \mathbb{N}$ points $\mathcal{D}^m \in \mathbb{R}^d$ for $m = 1, \ldots, \mathcal{M}$ is labelled with a binary label $p^m \in \{-1, 1\}$ for all $m = 1, \ldots, \mathcal{M}$; Hence,

$$\begin{cases} w^\top \phi(\mathcal{D}_m) + b \geq 0 & \text{if } p_m = +1 \\ w^\top \phi(\mathcal{D}_m) + b < 0 & \text{if } p_m = -1 \end{cases}$$

We can firstly implement a centralized Gradient Method to minimize a Logistic Regression Function to classify the previously generated points. The optimization problem becomes:

$$\min_{w,b} \sum_{m=1}^{\mathcal{M}} \log \Big(1 + \exp(-p^m (w^\top \phi(\mathcal{D}^m) + b)) \Big)$$

with $(w, b) \in \mathbb{R}^q \times \mathbb{R}$ as the optimization variable. We finally ran a set of simulations to test the effectiveness of our solution; we have tried different set-ups, including different dataset patterns and different separating functions. In the case of the ellipse, we assume the separating function to have 4 parameters, defining: $\phi(\mathcal{D}) := [|\mathcal{D}|_1, |\mathcal{D}|_2, (|\mathcal{D}|_1)^2, (|\mathcal{D}|_2)^2\,]^T$ We started to tackle the problem by building a classifier for linear boundaries and then scaling the difficulty to a parabola and finally to a ellipse, in this last algorithm we also introduced to possibility to insert in the training set some wrongly classified points to check the goodness of the algorithm.

1.2.1 Results

The following plots are obtained with this parameters choose by trial and error:

- $NumberOfPoints = 500$ training set size.
- $MaxIters = 10^4$ maximum number of iterations
- $\alpha = 10^{-2}$ constant step size.
- $TerminalCondition = 10^{-1}$ termination condition (10^{-6} if there are outlier points in the training set).
- $NumberOfTestPoints = 30\%$ of $NumberOfPoints(150)$ test set size.

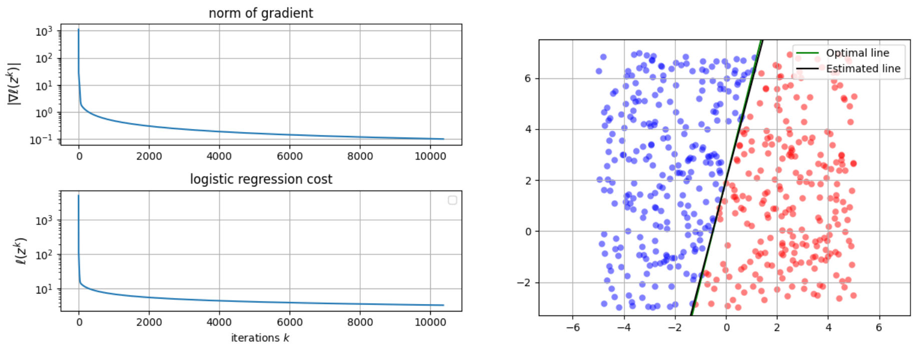


Figure 1.4: Cost function, gradient evolution and estimated linear classifier

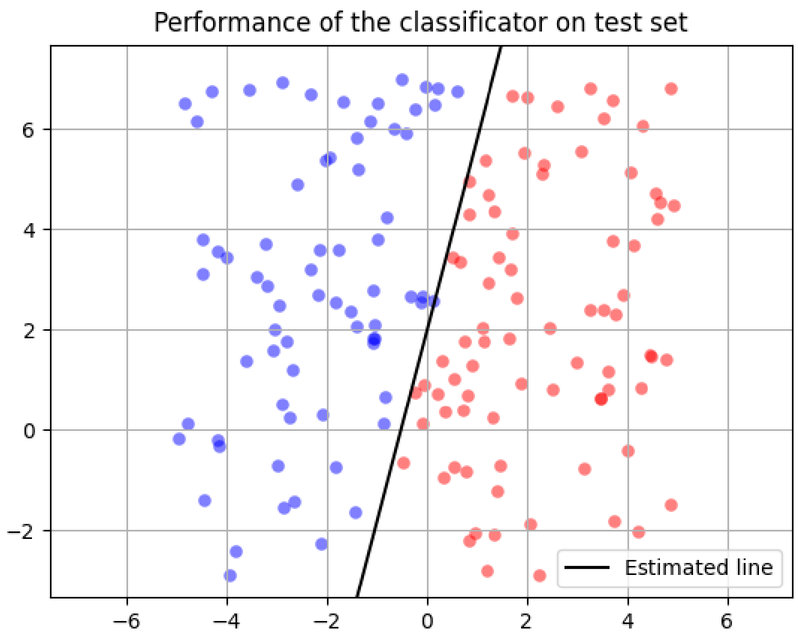


Figure 1.5: Result on test set of the obtained classifier (0 misclassified points)

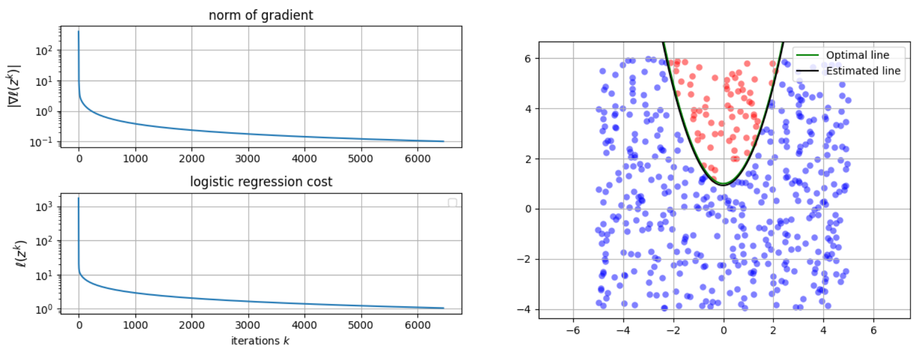


Figure 1.6: Cost function, gradient evolution and estimated parabola classifier

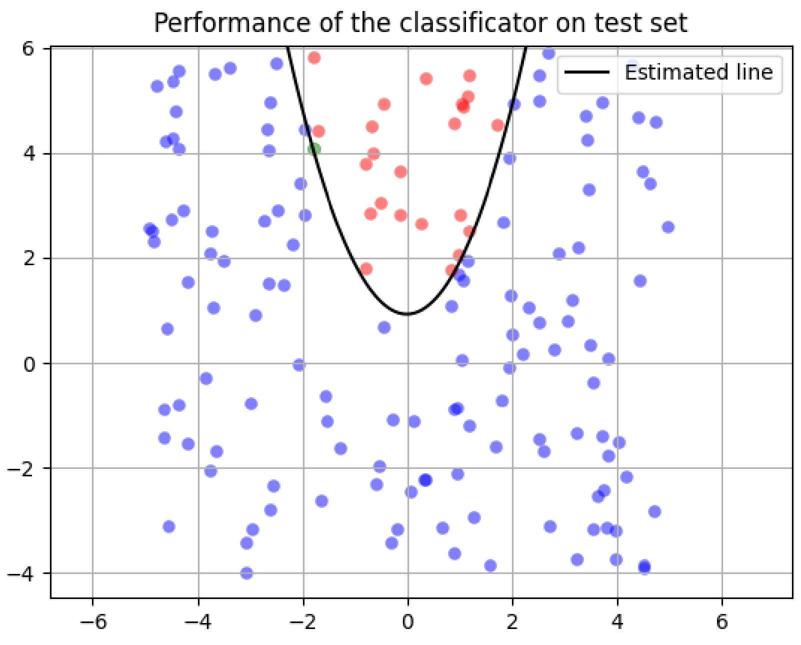


Figure 1.7: Result on test set of the obtained classifier (0 misclassified point)

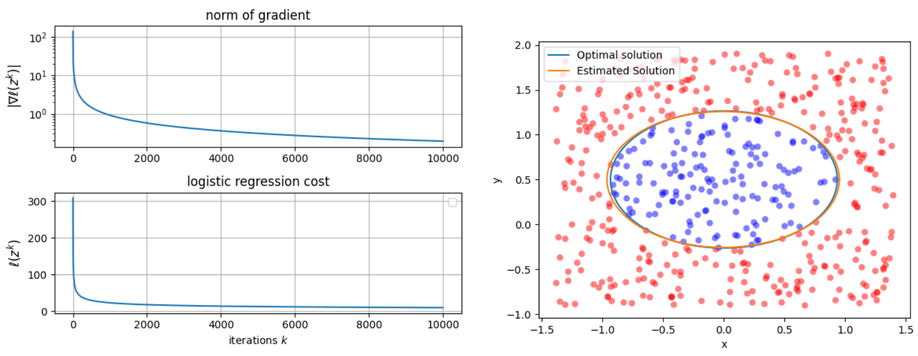


Figure 1.8: Cost function, gradient evolution and estimated parabola classifier

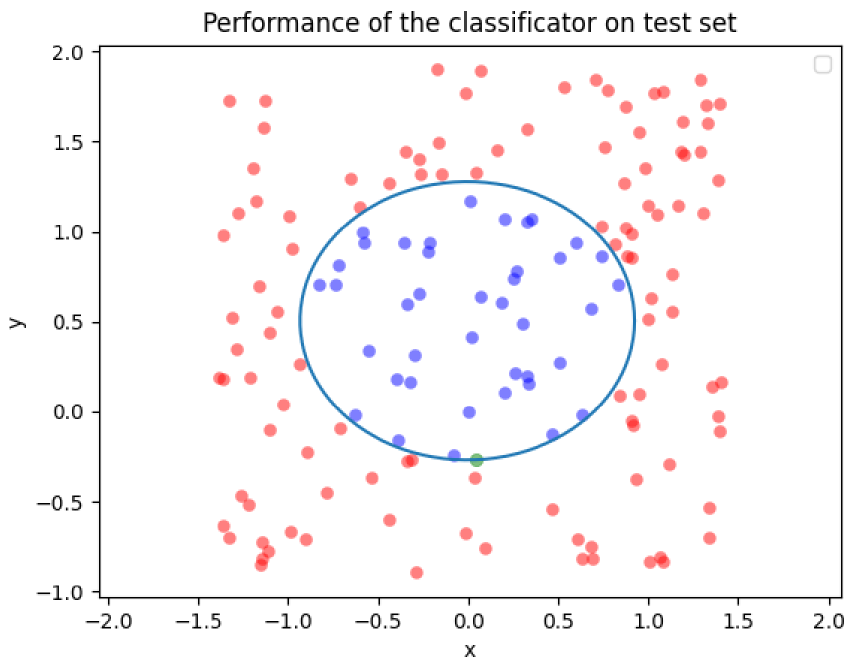


Figure 1.9: Result on test set of the obtained classifier (1 misclassified point)

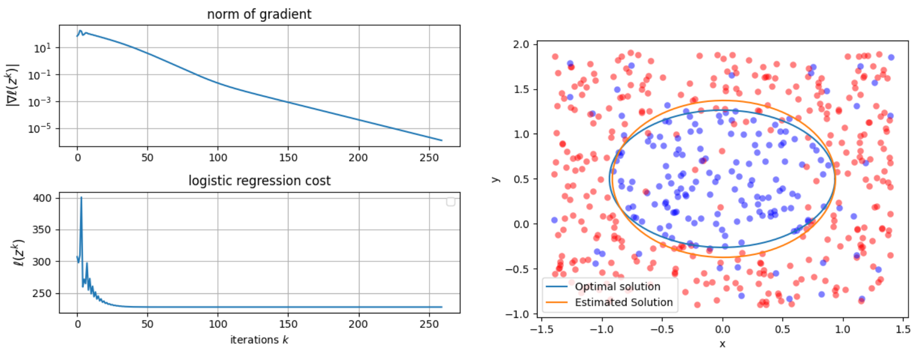


Figure 1.10: Cost function, gradient evolution and estimated parabola classifier

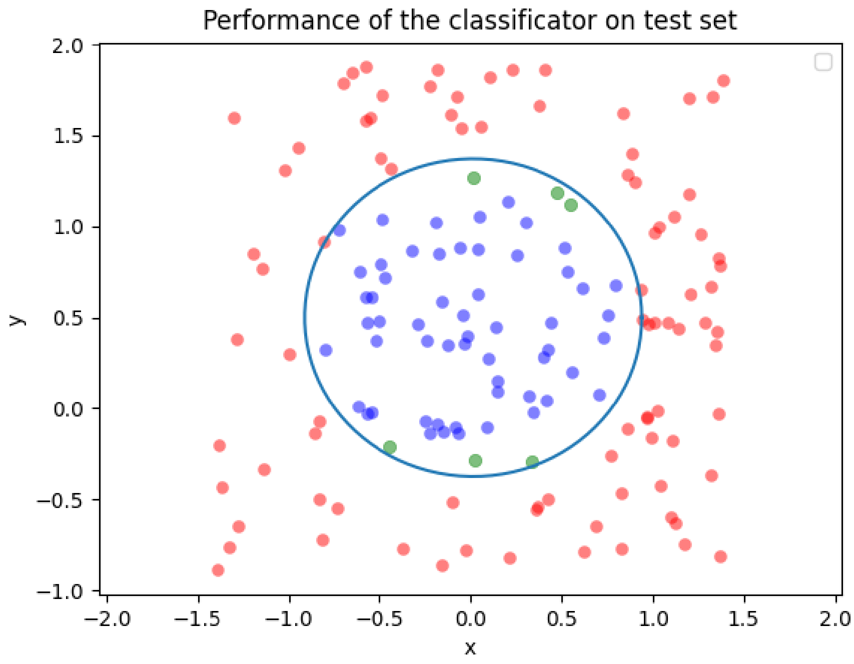


Figure 1.11: Result on test set of the obtained classifier (6 misclassified points)

1.3 Task 1.3: Distributed Classification

The problem now moves to a distributed setting. The dataset is randomly split between N subsets for each of the N agents, so that only a part of the whole is known by each, and not a single agent knows the whole structure of the set. The agents will work together and try to reach a consensus - which is, the nonlinear separating function defined in Task 1.2. We can now implement the Gradient Tracking algorithm designed in Task 1.1 to classify the dataset in a distributed fashion. The adjacency matrix $A_{ij} \in \mathbb{R}^{N \times N}$ is defined such that its elements a_{ij} are weighted by the Metropolis Hastings method:

$$A_{ij} = \begin{cases} \frac{1}{1+\max\{d_i,d_j\}} & \text{if } (i,j) \in E \text{ and } i \neq j \\ 1 - \sum_{h \in N_i \setminus \{i\}} A_{ih} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where:

- E is the set of edges in the graph.
- d_i is the degree of vertex i.
- N_i is the set of neighbors of vertex i.

We have run different simulations, testing different patterns and dataset sizes, showing how the distributed algorithm converges to a stationary point of the optimization problem. The reported one consists in 5 nodes and 150 points, with a star distribution pattern. We have also taken instants from different iterations to show the evolution of the learning of each node. The simulation runs for 2000 iterations.

We also compute the percentage of misclassified points as a quality test for our solution; then, we plot as usual the cost and the norm of its gradient.

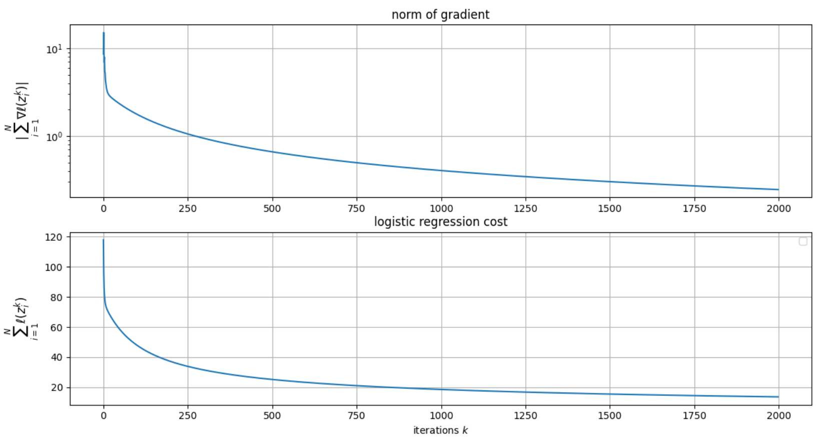


Figure 1.12: Gradient and Cost

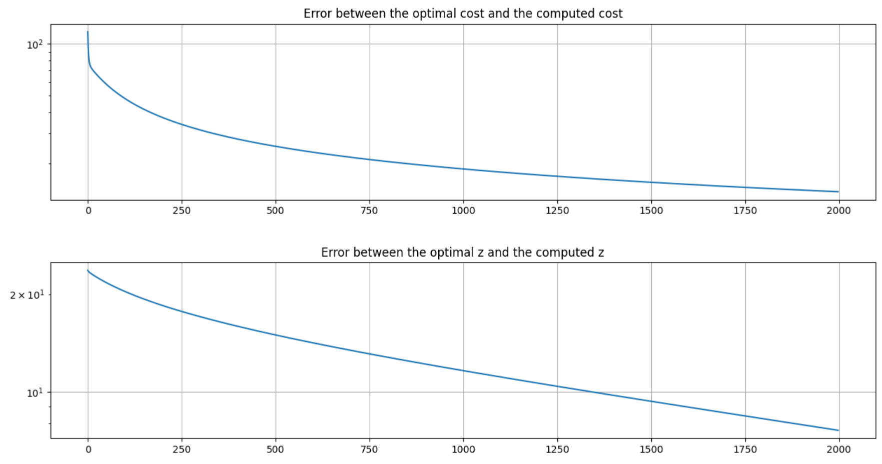
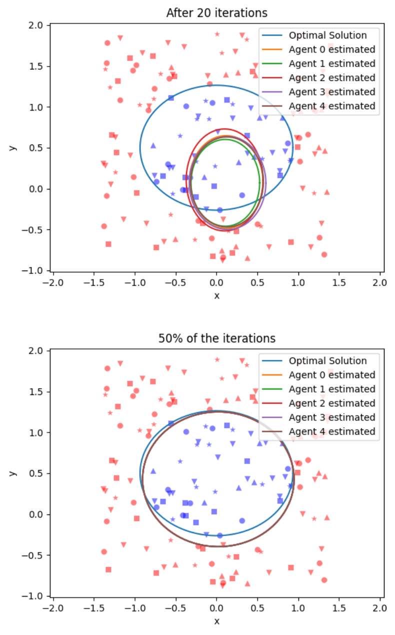


Figure 1.13: Error



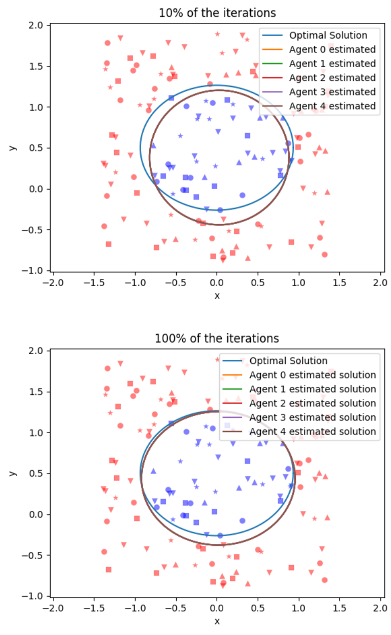


Figure 1.14: Simulation of a star graph with N = 5 and 150 points after 20, 200, 1000 and 2000 iterations.

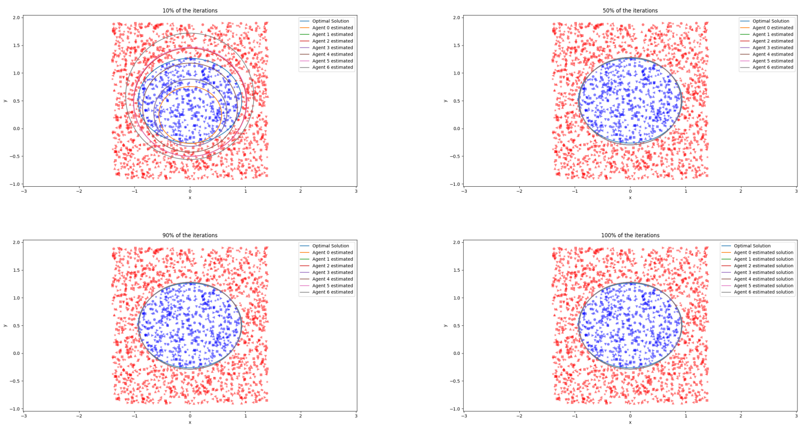


Figure 1.15: Simulating a binomial graph with N = 7 and 2000 points. Plots taken at 100, 500, 900 and 1000 iterations.

Chapter 2

Task 2: Aggregative Optimization for Multi-Robot Systems

Consider a team of N robots in a two-dimensional environment. Denote the position of robot $i \in \{1, \dots, N\}$ at iteration $k \in \mathbb{N}$ with $\mathbf{z}_i^k \in \mathbb{R}^2$ and denote with $\mathbf{z}^k \in \mathbb{R}^{2N}$ the stack vector of the locations of the whole team.

2.1 Task 2.1: Problem Set-Up

The goal of this task is to implement a distributed control algorithm that allows the robots to keep the formation tight, and, at the same time, be close to some private targets. This is formalized as an Aggregative Optimization problem:

$$\min_{z \in \mathbb{R}^d} \sum_{i=1}^{N} \ell_i(z_i, \sigma(z)), \quad \sigma(z) \equiv \frac{\sum_{i=1}^{N} \phi_i(z_i)}{N}$$

where $\ell_i(z_i, \sigma(z)) : \mathbb{R}^2 \times \mathbb{R}^d \to \mathbb{R}$, $\phi_i(z_i) : \mathbb{R}^2 \to \mathbb{R}^d$. Here, each agent i is only aware of its respective decision variable and functions z_i, ℓ_i, ϕ_i. The cost function is defined as $\ell_i(z_i, \sigma(z)) = \gamma_i \|z_i - r_i\|^2 + \|\sigma(z) - r_0\|^2$. γ_i is a trade-off variable, defining which term between the two has more importance in the function. The higher, the closer the agents will be to their targets; the lower, the closer the agents will be to their barycenter The global aggregative variable $\sigma(z)$ is the barycenter of the agents' team: hence, $\sigma_i(z_i) = \frac{1}{N} \sum_{i=1}^{N} z_i \quad \forall i = 1, \dots, N$. We implemented a version of the Aggregative Tracking algorithm to satisfy the requirements of the task mentioned before. To do so we've kept track of the estimates of the decision variable z, of the barycenter σ and of the gradient in σ of the cost at each iteration k to obtain:

$$\begin{cases} z_i^{k+1} = z_i^k - \alpha(\nabla_1 \ell_i(z_i^k, s_i^k) + \nabla\phi_i(z_i^k)v_i^k) & z_i^0 \in \mathcal{R}^{n_i} \\ s_i^{k+1} = \sum_{j\in N_i} a_{ij}s_j^k + \phi_i(z_i^{k+1}) - \phi_i(z_i^k) & s_i^0 = \phi_i(z_i^0) \\ v_i^{k+1} = \sum_{j\in N_i} a_{ij}v_j^k + \nabla_2\ell_i(z_i^{k+1}, s_i^{k+1}) - \nabla_2\ell_i(z_i^k, s_i^k) & v_i^0 = \nabla_2\ell_i(z_i^0, s_i^0) \end{cases}$$

We have run different simulations to show how the algorithm performs with different tuning parameters and target locations. We have also tried testing a configuration in which targets move to observe how the agents would react and how the cost is changing accordingly. While the targets are moving, the cost function cannot find a minimum because the setup of the problem is still evolving and changing, not until we stop them at least. To avoid this behaviour an integral action could be used.

2.1.1 Results

The following plots are obtained with this parameters choose by trial and error:

- $N = 4$ number of agents.
- $MaxIters = 2000$ maximum number of iterations
- $\alpha = 10^{-3}$ constant step size.
- $d = 2$ dimension of the agent's state.
- $\gamma = 10$ trade-off parameter that practically sets how much every agent will be close to its target.

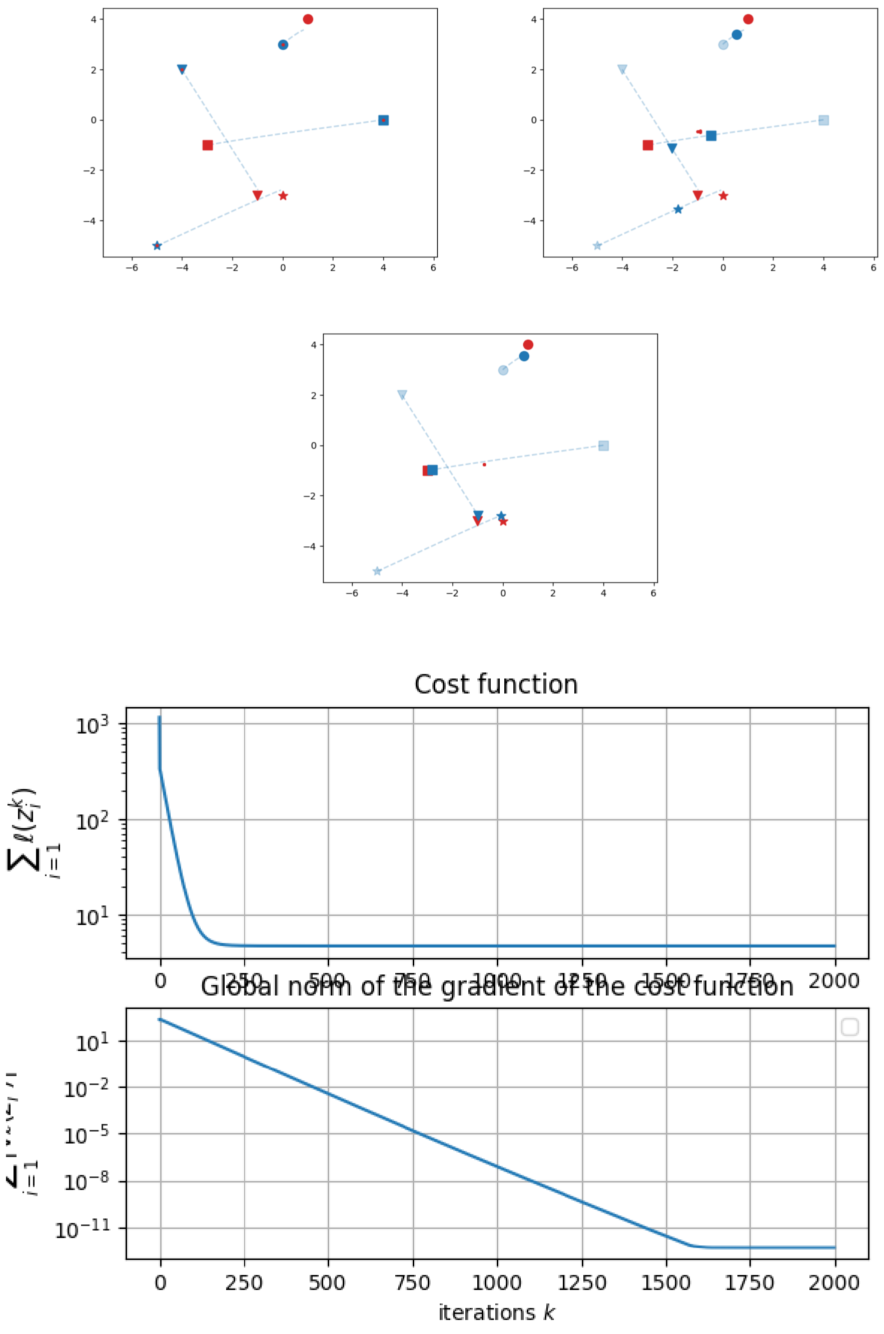


Figure 2.1: Evolution of the position of the agents with respect to static targets, evolution of the cost function and of the gradient of the cost function.

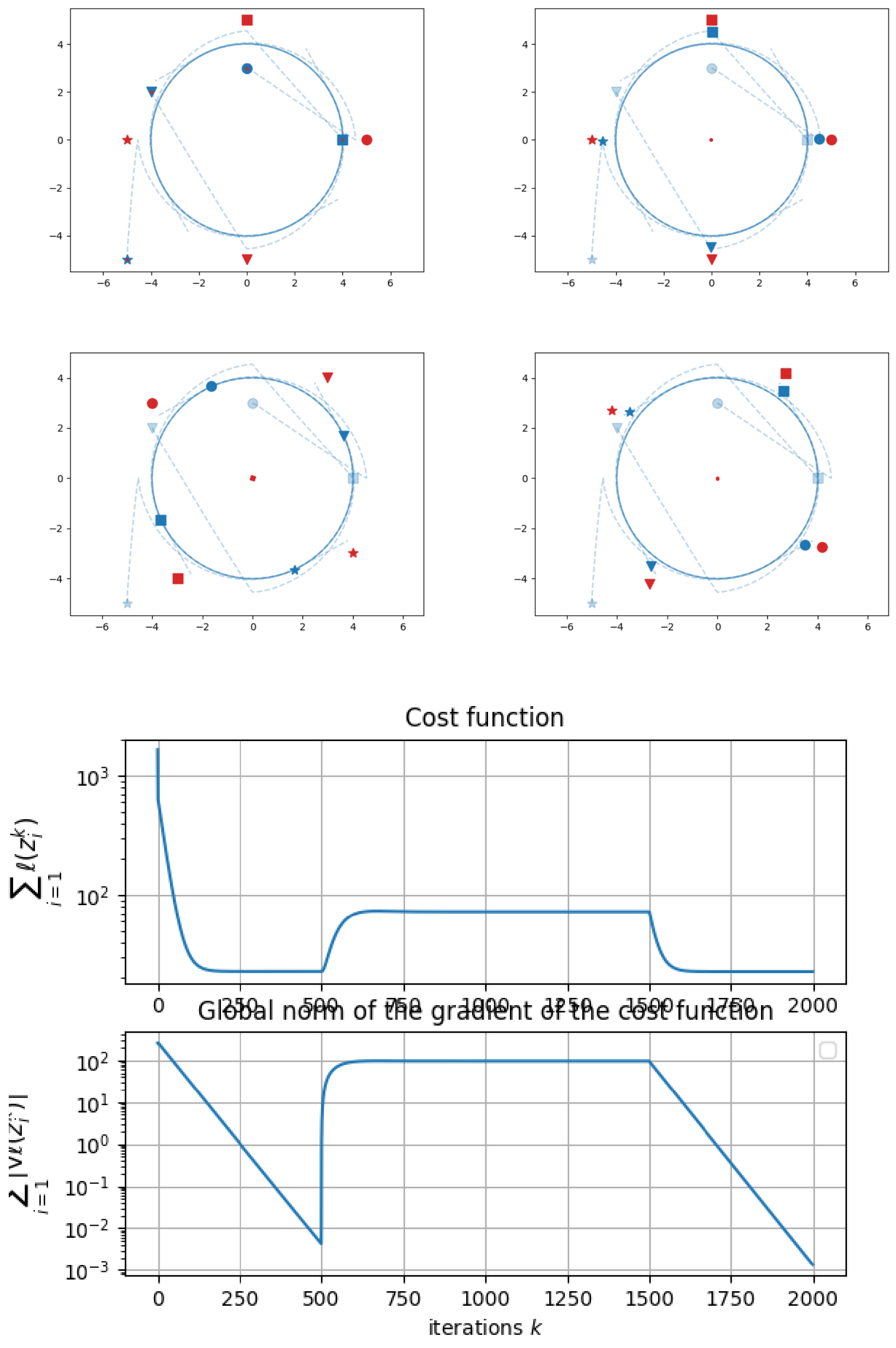


Figure 2.2: Evolution of the position of the agents with respect to targets moving in a circular pattern and then stopping, evolution of the cost function and of the gradient of the cost function.

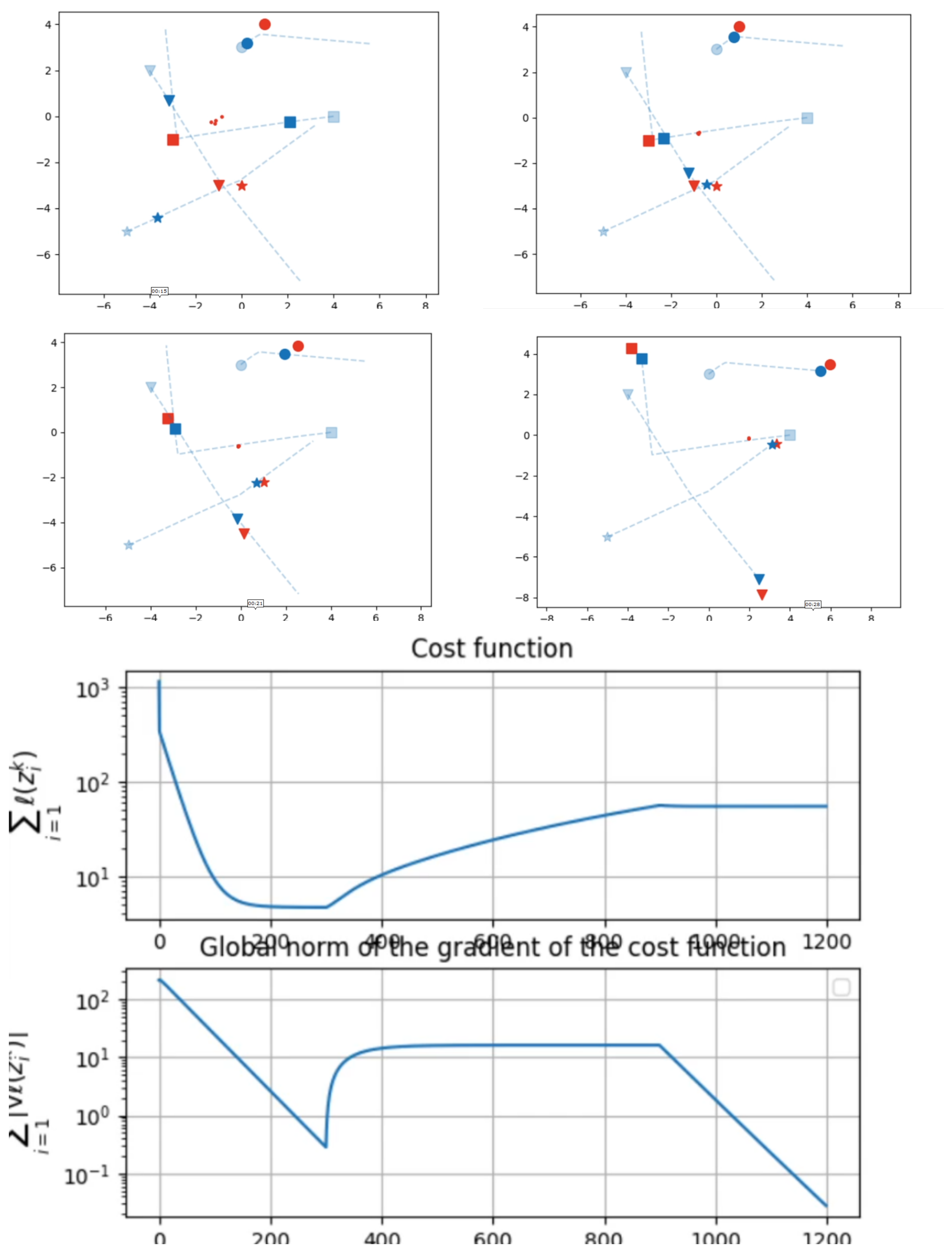


Figure 2.3: Evolution of the position of the agents with respect to targets moving in a random pattern and then stopping, evolution of the cost function and of the gradient of the cost function.

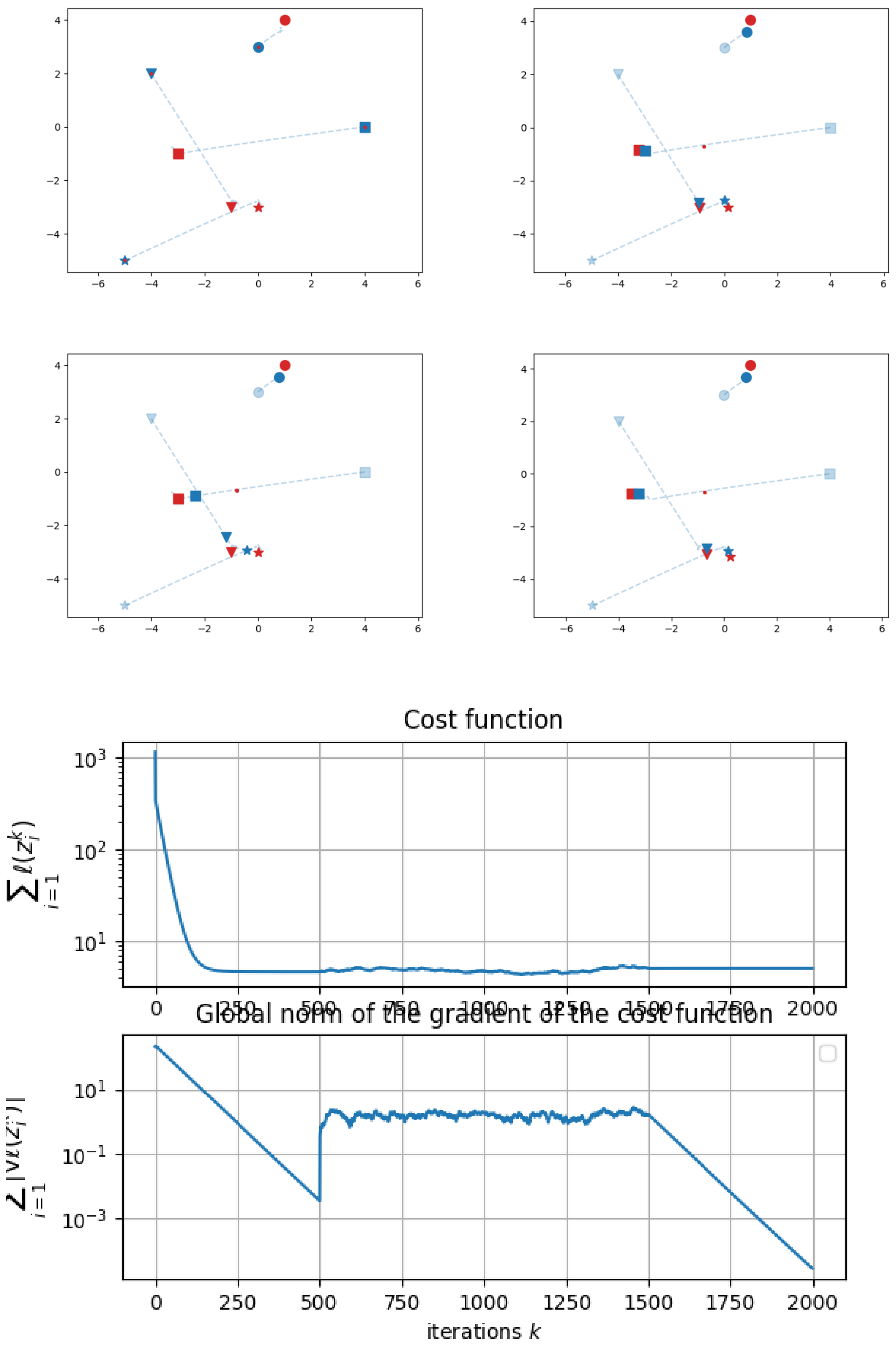


Figure 2.4: Evolution of the position of the agents with respect to targets moving in a random direction, evolution of the cost function and of the gradient of the cost function.

2.2 Task 2.2: ROS 2 Implementation

After having created a ROS 2 package, we implemented our Aggregative Tracking algorithm from 2.1 to run and be displayed on RViz. ROS 2 enables the creation of nodes that can publish messages to and receive them from other nodes; we have written our code in such a fashion as to synchronize the communications at each iteration k with a barrier. The nodes are Objects containing parameters from the launch file, such as the dimension d, the number of agents N, the graph structure and its neighbours, but also the estimates $z_i^{k+1}, s_i^{k+1}, v_i^{k+1}$, their initial values z_i^0, s_i^0, v_i^0 and the indexes used to store them in $\mathbf{z}^k, \mathbf{r}^k \in \mathbb{R}^{2N}$. The weighted adjacency matrix is created with Metropolis-Hastings weights. An additional *plotter* node is launched to ensure the positions of the agents, the targets and the barycenter are visualized correctly in RViz. Agents are displayed as green spheres, Targets as red cubes and Barycenters as red dots.

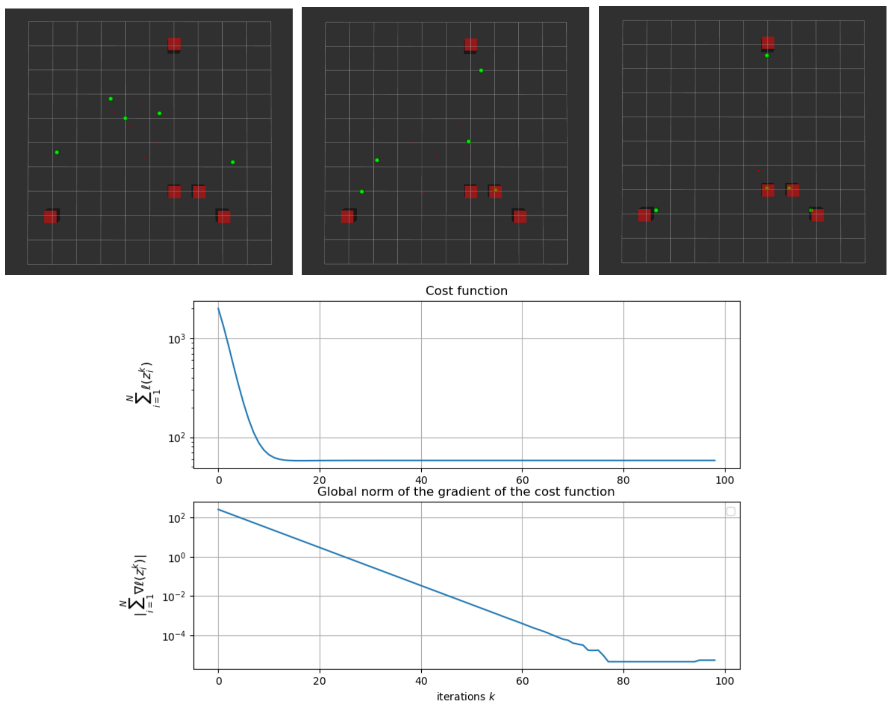


Figure 2.6: Binomial Graph, N = 5, $\gamma = 10$. Evolution in time from start to finish in 100 iterations, cost and gradient.

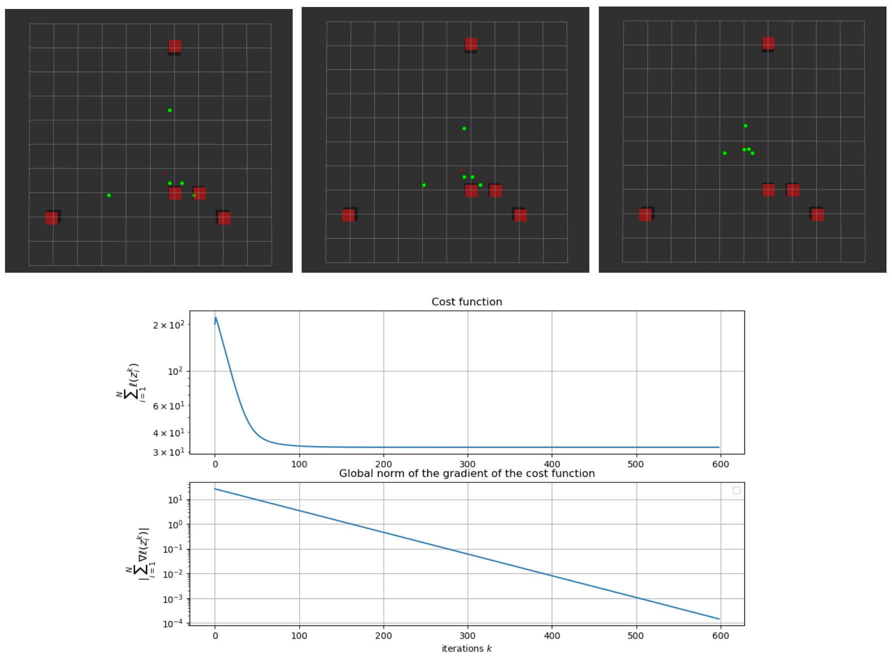


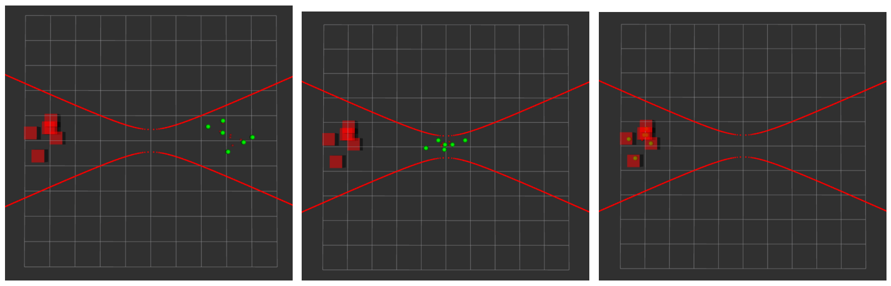
Figure 2.7: Different Tradeoff values: $\gamma = 1$, 0.5, 0.2. Ending configurations after 600 iterations.

2.3 Task 2.3: Moving inside a corridor

The task now requires agents to enter a corridor, without colliding with its walls, and then reach some targets at the end of it, all so while keeping the formation tight. This is done by adding a barrier function to the cost:

$$-\epsilon \log(-5x^2 + y^2 + 1),\ \epsilon > 0$$

The barrier is an hyperbole and is shown in RViz as well by means of the *plotter* node. We then plot the cost and the norm of its gradient.



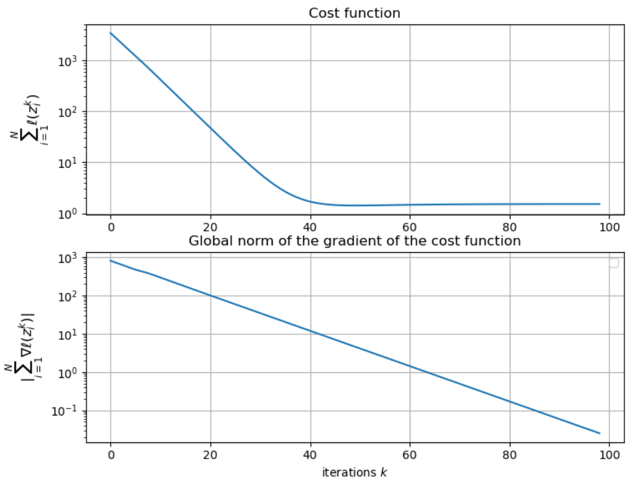

Cost function
$\sum_{i=1}^{N} \ell(z_i^k)$
10^0
10^1
10^2
10^3
0
20
40
60
80
100
Global norm of the gradient of the cost function
$\|\sum_{i=1}^{N} \nabla \ell(z_i^k)\|$
10^{-1}
10^0
10^1
10^2
10^3
0
20
40
60
80
100
iterations k

Figure 2.8: Simulation with barrier function; N = 5, γ = 10. Snapshots across 200 iterations.

Conclusions

In this report, we showed a possible implementation to build a distributed classifier based on a logistic function, tested its performance with a test set and showed how different graph connections and data sets behaved on our solution. We then built an aggregative optimization algorithm to control a multi-robot system to move towards some targets while maintaining the formation tight, and simulated on ROS with different tradeoff parameters and cost functions. The videos of all the animations in RViz are available in the shared folder.

Bibliography

[1] H. K. Khalil, *Nonlinear Systems*, 1996.

[2] Francesco Bullo, *Lectures on Network Systems*, 2022.

[3] Giuseppe Notarstefano, Ivano Notarnicola, *Lecture Notes on Distributed Autonomous System*, 2024.